Noise-contrastive estimation of normalising constants and GANs

Contents

1 Fonctions génériques

```
library(ggplot2)
library(reshape)
library(matrixStats)
```

1.1 Algorithme d'Hasting

Utilité : simuler selon $p_m(., \psi)$ pour un paramètre ψ choisi.

Argument	Type	Exemple	Indication
x	vecteur	rcauchy(100, 0, 1)	notre échantillon de densité inconnue
n	entier	100	taille de la simulation
psi	vecteur	c(0,1)	paramètres de la fonction h
h	fonction		fonction qui retourne $\overline{p_m}(., \psi)$

```
hasting = function(x, n, psi, h){
  y = c()
  y = append(y, sample(sample(x, 1)))
  for (i in 2:n){
    y_ = rnorm(1, y[i-1], 1)
    u = runif(1)
    if ( u <=
         (h(y_,psi) * dnorm(y_, y[i-1], 1))
         /(h(y[i-1],psi) * dnorm(y[i-1], y_, 1))
    ){
      y = append(y, y_)}
    else {
      y = append(y, y[i-1])
    }
  }
  return (y)
}
```

Note : on peut très certainement écrire sous forme matricielle cette fonction pour une meilleure performance.

1.2 MC MLE

Utilité : retourne une estimation des paramètres selon la méthode décrite dans le papier de Geyer.

```
mc_mle = function(x, psi, h){

  m = length(x)

  y = hasting(x, m, psi, h)

  L = function(theta){
    return(sum(log(h(x,theta)/h(x,psi))) - m*log(mean(h(y,theta)/h(y,psi))))
  }

  theta = optim(
    par = rep(1,length(psi)),
    gr = "CG",
    control = list(fnscale=-1),
```

```
    fn = L
  )$par

  return(theta)
}
```

1.3 NCE

Utilité : Retourne l'estimation de la constante et des paramètres.

Argument	Type	Exemple	Indication
x	vecteur	rcauchy(100, 0, 1)	notre échantillon de densité inconnue
law_y	fonction	rnorm	fonction qui retourne un échantillon suivant la loi p_n
n	entier	100	taille de l'échantillon de bruit suivant la loi p_n
params_y	vecteur	c(0,1)	arguments de la fonction law_y
log_pm	fonction		fonction qui retourne le logarithme de la densité p_m
log_pn	fonction		fonction qui retourne le logarithme de la densité p_n
size_theta	entier	3	taille de θ, vaut habituellement 2 ou 3
method	string	"CG"	méthode d'optimisation, habituellement "CG" ou "BFGS"

```
nce = function(x, law_y, params_y, log_pm, log_pn, size_theta, n){

  y = do.call(law_y, c(list(n),params_y))

  m = length(x)

  h = function(u, theta){
    return( 1 / (1 + n/m * exp(log_pn(u) - log_pm(u, theta))))
  }

  J = function(theta){
    return( sum(log(h(x, theta))) + sum(log(1 - h(y, theta))) )
  }

  theta = optim(
    par = rep(1, size_theta),
    gr = "CG",
    control = list(fnscale=-1),
    fn = J
  )$par

  return(c(theta[-size_theta], exp(-theta[size_theta])))
}
```

1.4 Graphiques

Utilité : afficher l'histogramme pour un échantillon de données x.

```
print_hist = function(x) {
  df = data.frame(x = x)
  hist_x = ggplot(df, aes(x=x)) +
            geom_histogram(aes(y = stat(count)/sum(count)), bins = 20, color="white") +
            theme(aspect.ratio = 1) +
            labs(y = "Fréquence") +
```

```
          ggtitle("Distribution de l'échantillon x")
  print(hist_x)
}
```

Utilité : pour NCE, afficher l'évolution des paramètres au fur et à mesure de l'augmentation de n (la dimension de l'échantillon de bruit)

```
NCE_evol_params = function(x, law_y, params_y, log_pm, log_pn, size_theta, ratio, steps, labels) {

  # Creation de l'abscisse
  m = length(x)
  N = seq(0, m*ratio, length.out = steps + 1)

  # Creation de l'ordonnée
  theta = c()
  for (n in N) {
    theta = append(theta, nce(x, law_y, params_y, log_pm, log_pn, size_theta, n))
  }

  # Formatage des données
  theta = t(rbind(matrix(theta, nrow = size_theta),N))
  df = as.data.frame(theta)
  df_melted = melt(df, id.vars = "N")

  # Plot
  plot_df = ggplot(df_melted, aes(x = N, y = value)) +
  geom_line(aes(color = variable, group = variable)) +
  geom_point(aes(color = variable, group = variable)) +
  labs(title = "Evolution des paramètres par rapport au bruit",
       x = "n (taille du bruit)",
       y = "Paramètres",
       color = "Légende") +
  scale_color_manual(labels = labels, values = c("blue", "red", "orange"))

  print(plot_df)

  #return(theta)
}
```

Note : il faudrait optimiser le temps de calcul de ces fonctions, peut-être en matriciel au lieu des boucles ou bien avec du calcul en parralèle sur CPU/GPU

2 Applications

2.1 Exemple basique : la loi normale

Soit x l'échantillon de taille m obtenu selon la loi de densité inconnue p_d.

On considère ici que p_d appartient à la famille de fonctions paramétrées par $\theta = (c, \mu, \sigma)$ suivante :

$$p_m(u;\theta) = \frac{1}{Z(\mu,\sigma)} \times exp\big[-\frac{1}{2}\big(\frac{u-\mu}{\sigma}\big)^2\big] \quad \text{d'où} \quad ln(p_m(u;\theta)) = c - \frac{1}{2}\big(\frac{u}{\sigma} - \frac{\mu}{\sigma}\big)^2$$

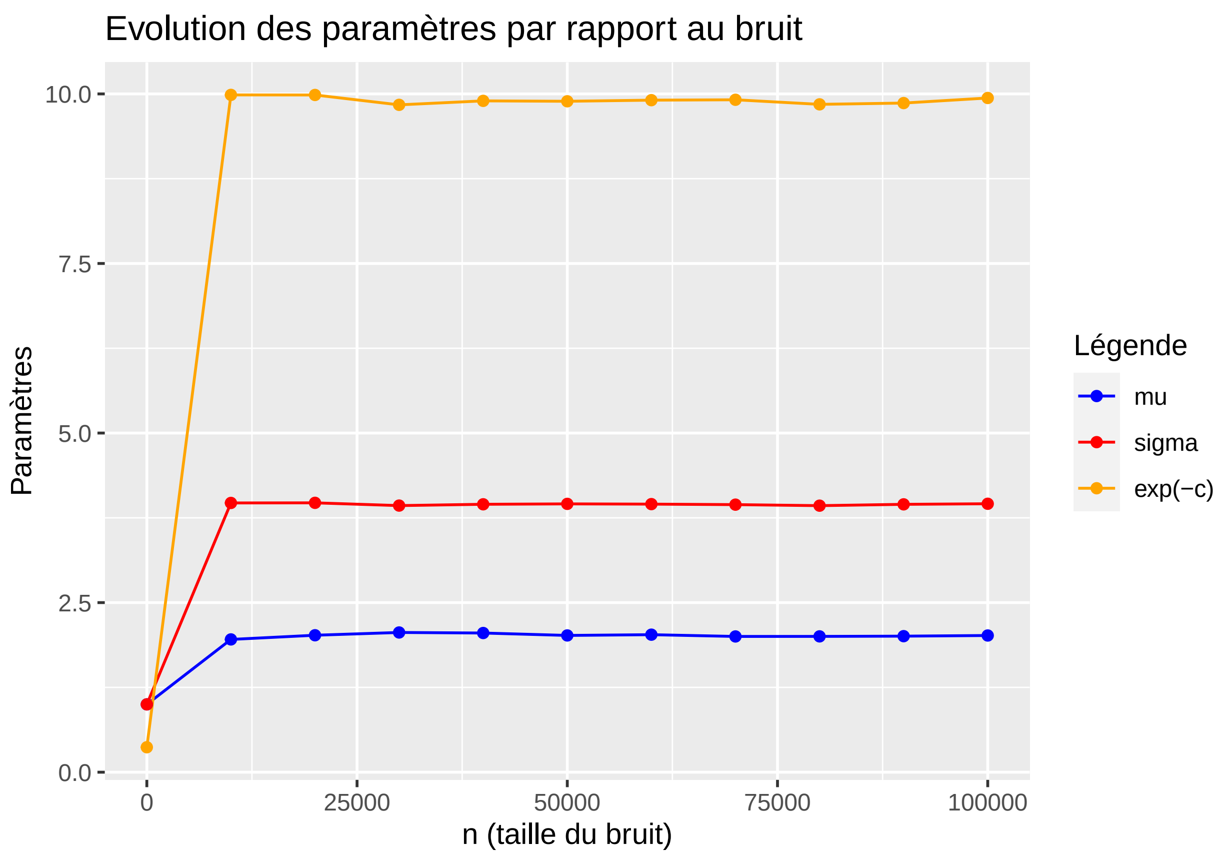
```
pm_barre = function(u, theta){
  return(exp(-0.5 * ((u - theta[1]) / theta[2]) ** 2))
}

log_pm = function(u,theta){
  return(theta[3] - 1/2 * (u/theta[2] - theta[1]/theta[2]) ** 2)
  # theta[1] = mu / theta[2] = sigma / theta[3] = c
}

log_pn_cauchy = function(u){
  return(log(dcauchy(u, mean(x), sd(x))))
}

m = 10000
n = 10000
x = rnorm(m, 2, 4)
size_theta = 3

# METHODE MC MLE
mc_mle(x, c(mean(x),sd(x)), pm_barre)
```

```
## [1] 1.739094 4.272469
```

```
# METHODE GEYER
nce(x, rcauchy, c(mean(x),sd(x)), log_pm, log_pn_cauchy, size_theta, n)
```

```
## [1] 2.048061 3.944253 9.871508
```

```
NCE_evol_params(x, rcauchy, c(mean(x),sd(x)), log_pm, log_pn_cauchy, size_theta, 10, 10,
                c("mu", "sigma", "exp(-c)"))
```



Evolution des paramètres par rapport au bruit
Paramètres
10.0
7.5
5.0
2.5
0.0
0
25000
50000
75000
100000
n (taille du bruit)
Légende
mu
sigma
exp(−c)

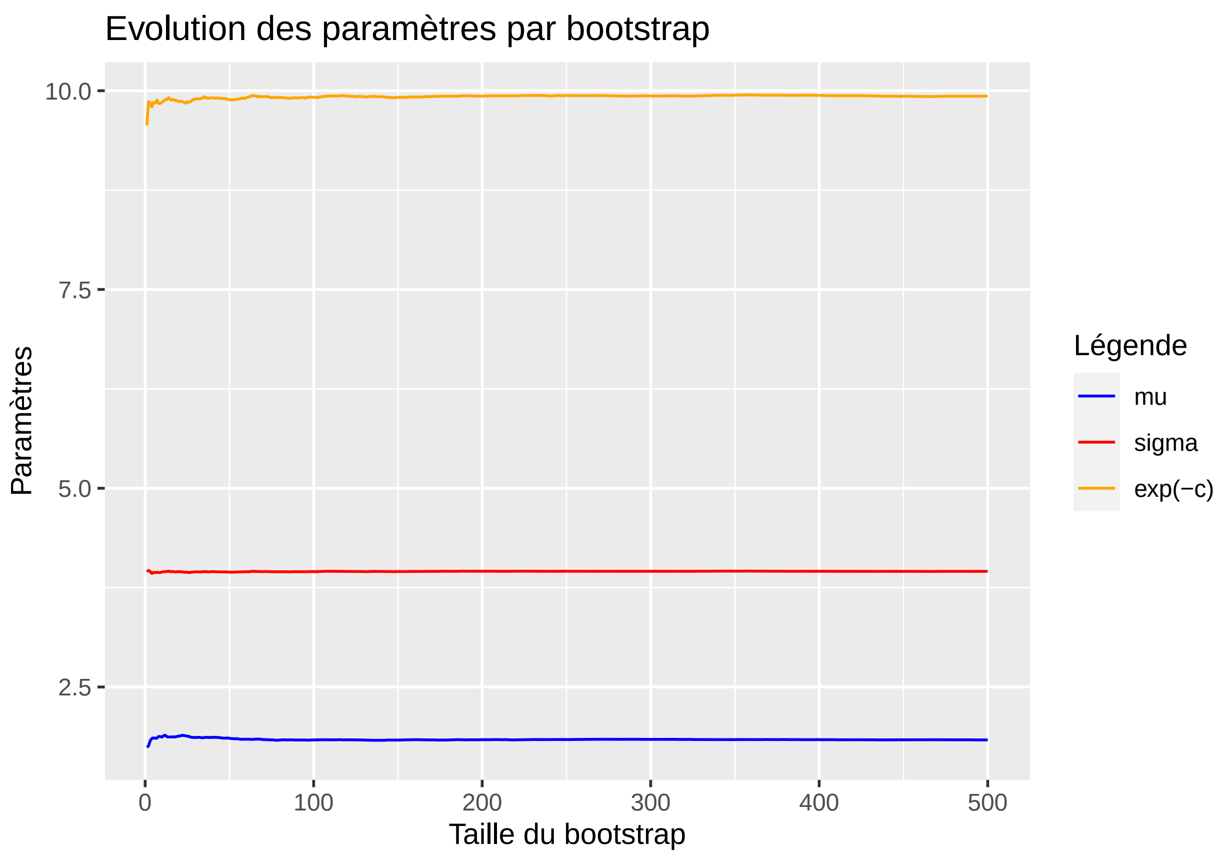
3 Nouvelles approches

3.1 Bootstrap

Utilité : utilise le bootstrap sur x pour estimer les paramètres

```
NCE_bootstrap = function(x, law_y, params_y, log_pm, log_pn, size_theta, n, size_boot, labels) {

  theta_bootstrap = c()

  for (i in 1:size_boot) {
    x_bootstrap = sample(x, size = m, replace=TRUE)
    theta_bootstrap = append(theta_bootstrap, nce(x_bootstrap,
                                                  law_y,
                                                  params_y,
                                                  log_pm,
                                                  log_pn,
                                                  size_theta,
                                                  n))
  }

  # Formatage des données
  theta_bootstrap = matrix(theta_bootstrap, nrow = size_theta)

  # Plot
  array_boot = 1:size_boot
  df = as.data.frame(cbind(t(rowCumsums(theta_bootstrap))/array_boot, array_boot))
  df_melted = melt(df, id.vars = "array_boot")

  plot_df = ggplot(df_melted, aes(x = array_boot, y = value)) +
  geom_line(aes(color = variable, group = variable)) +
  labs(title = "Evolution des paramètres par bootstrap",
       x = "Taille du bootstrap",
       y = "Paramètres",
       color = "Légende") +
  scale_color_manual(labels = labels, values = c("blue", "red", "orange"))

  print(plot_df)

  return(rowMeans(theta_bootstrap))
}
```

```
NCE_bootstrap(rnorm(1000,2,4), rcauchy, c(mean(x),sd(x)), log_pm, log_pn_cauchy, size_theta, 1000, 500,
```




```
## [1] 1.834298 3.954897 9.932777
```

3.2 Récursivité

Utilité : améliorer récursivement la précision de l'estimation via les estimations précédentes

```
mc_mle_recursif = function(x, psi, h, size_of_loop){

  m = length(x)

  for (i in 1:size_of_loop) {
    y = hasting(x, m, psi, h)

    L = function(theta){
      return(sum(log(h(x,theta)/h(x,psi))) - m*log(mean(h(y,theta)/h(y,psi))))
    }

    psi = optim(
      par = rep(1,length(psi)),
      gr = "CG",
      control = list(fnscale=-1),
      fn = L
    )$par

    print(psi)
  }
```

```
  return(psi)
}
```

```
mc_mle_recursif(x, c(mean(x),sd(x)), pm_barre, 10)
```

```
## [1] 1.576203 3.714840
## [1] 1.635573 4.230301
## [1] 2.139895 3.978243
## [1] 1.768109 4.100622
## [1] 1.380575 3.933015
## [1] 2.217360 3.760074
## [1] 1.829862 4.514500
## [1] 1.636047 3.977075
## [1] 1.734133 3.877315
## [1] 2.263114 3.773557
```

```
## [1] 2.263114 3.773557
```

```
mc_mle_recursif_2 = function(x, psi, h, size_of_loop){

  m = length(x)
  M = m

  for (i in 1:size_of_loop) {
    y = hasting(x, M, psi, h)

    L = function(theta){
      return(sum(log(h(x,theta)/h(x,psi))) - M*log(mean(h(y,theta)/h(y,psi))))
    }

    theta = optim(
      par = rep(1,length(psi)),
      gr = "CG",
      control = list(fnscale=-1),
      fn = L
    )$par

    x = append(x, hasting(x, m, theta, h))
    M = M + m

    print(theta)
  }

  return(theta)
}
```

```
mc_mle_recursif_2(x, c(mean(x),sd(x)), pm_barre, 10)
```

```
## [1] 1.979904 3.842944
## [1] 1.895677 4.196975
## [1] 1.649561 3.812036
## [1] 2.189071 3.792279
## [1] 1.894939 3.894201
## [1] 1.929402 3.811057
## [1] 1.928945 3.891135
## [1] 1.777666 3.775692
```

```
## [1] 2.025873 3.875512
## [1] 2.065568 3.812067

## [1] 2.065568 3.812067
```

```
mc_mle_recursif_3 = function(x, psi, h, size_of_loop){

  m = length(x)
  M = m
  y = hasting(x, m, psi, h)

  for (i in 1:size_of_loop) {

    L = function(theta){
      return(sum(log(h(x,theta)/h(x,psi))) - M*log(mean(h(y,theta)/h(y,psi))))
    }

    theta = optim(
      par = rep(1,length(psi)),
      gr = "CG",
      control = list(fnscale=-1),
      fn = L
    )$par

    x = append(x, hasting(x, m, theta, h))
    y = append(y, hasting(x, m, psi, h))
    M = M + m

    print(theta)
  }

  return(theta)
}
```

```
mc_mle_recursif_3(x, c(mean(x),sd(x)), pm_barre, 10)
```

```
## [1] 1.891314 3.733184
## [1] 1.767275 3.701550
## [1] 1.798429 3.581107
## [1] 1.766843 3.563097
## [1] 1.777424 3.541060
## [1] 1.773666 3.570075
## [1] 1.779948 3.577016
## [1] 1.787526 3.568270
## [1] 1.844896 3.570151
## [1] 1.894364 3.564316

## [1] 1.894364 3.564316
```

```
nce_recursif = function(x, law_y, params_y, log_pm, log_pn, size_theta, n, size_of_loop){

  y = do.call(law_y, c(list(n),params_y))

  m = length(x)

  for (i in 1:size_of_loop){
```

```
    h = function(u, theta){
      return( 1 / (1 + n/m * exp(log_pn(u) - log_pm(u, theta))))
    }

    J = function(theta){
      return( sum(log(h(x, theta))) + sum(log(1 - h(y, theta))) )
    }

    theta = optim(
      par = rep(1, size_theta),
      gr = "CG",
      control = list(fnscale=-1),
      fn = J
    )$par

    print(theta)

    y = do.call(law_y, c(list(n),theta[-size_theta]))

  }

  return(c(theta[-size_theta], exp(-theta[size_theta])))
}
```

```
nce_recursif(x, rcauchy, c(mean(x),sd(x)), log_pm, log_pn_cauchy, size_theta, n, 10)
```

```
## [1]  2.036572  3.939894 -2.296987
## [1]  2.076766  3.930706 -2.281932
## [1]  1.992583  3.969417 -2.304411
## [1]  2.051985  3.943557 -2.299769
## [1]  1.996039  3.945949 -2.294777
## [1]  2.029782  3.969944 -2.296824
## [1]  1.965352  3.959583 -2.294070
## [1]  1.997492  3.929059 -2.280146
## [1]  2.026969  3.947495 -2.295299
## [1]  2.010646  3.974863 -2.296041
```

```
## [1] 2.010646 3.974863 9.934770
```